# Trabalho Parcial I
# Módulo ondeestou

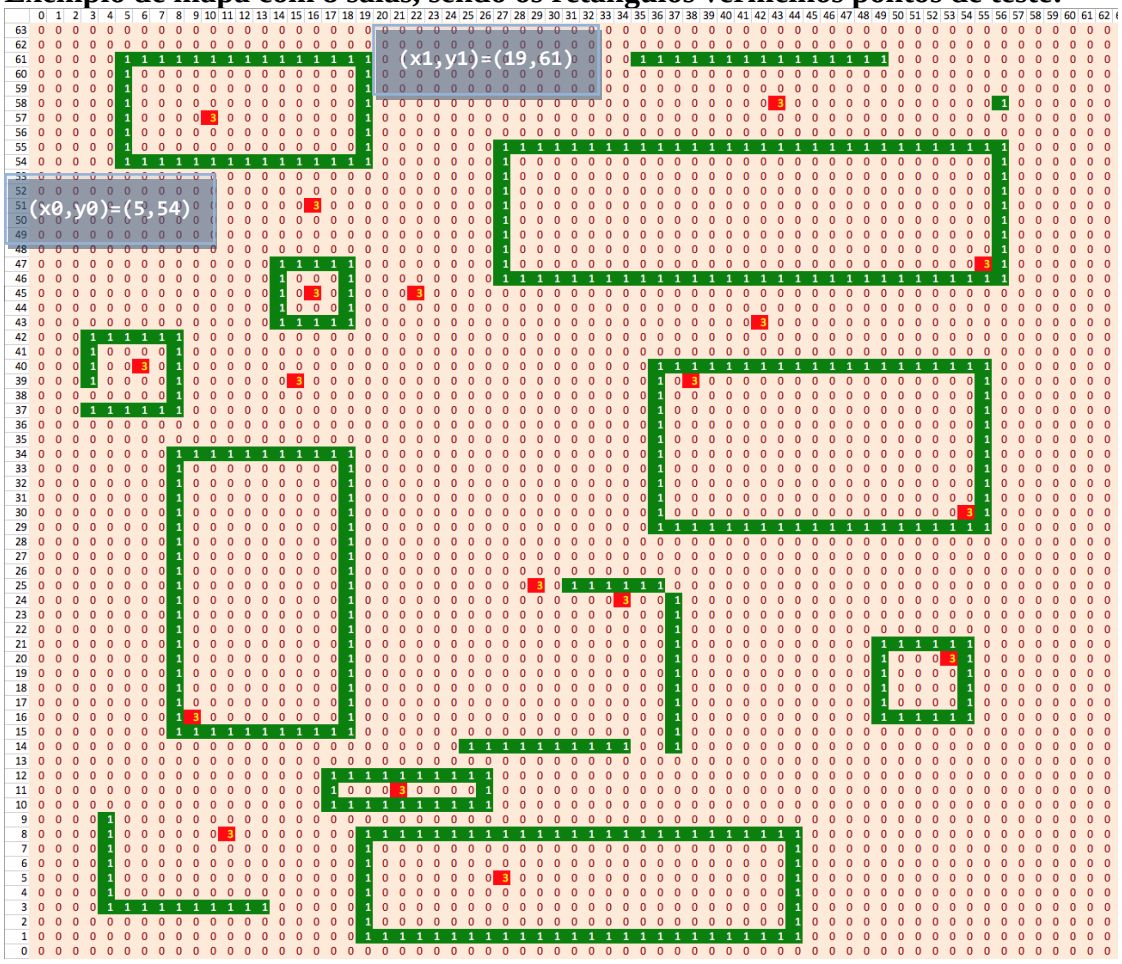
*Baseado em material gentilmente fornecido pelo Prof. Dr. Fernando Moraes*

**Agosto/2023**

# Objetivo: desenvolvimento de máquina de estados

O circuito a ser desenvolvido tem por função localizar em um dado **mapa** se a posição de um dado dispositivo (por exemplo, um robô) encontra-se em uma determinada sala. O mapa possui uma série de salas, e o circuito deve informar se a posição fornecida está dentro ou não de uma dada sala. O circuito também deverá informar em qual sala o dispositivo está localizado, a partir do tamanho de cada sala.

**Exemplo de mapa com 8 salas, sendo os retângulos vermelhos pontos de teste:**




**A coordenada inferior tem valor $y = 0$ e a coordenada superior valor $y = 63$.**

**Observação:**
No VHDL o mapa está invertido. Esta inversão deve-se apenas como está **declarado o VHDL.**

**Em destaque as coordenadas de canto de uma sala $[(x0, y0), (x1, y1)]$.**

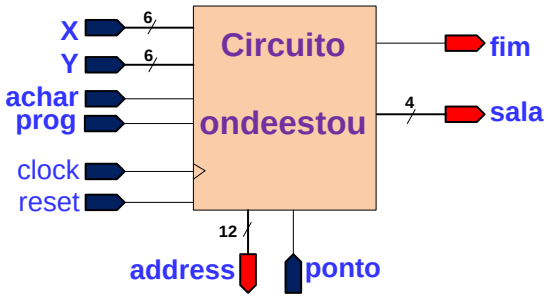
Restrições

- Todas as salas tem 4 lados, e as linhas formando a sala são contínuas
- Os pontos a serem avaliados (vermelhos) não devem ficar sobre as bordas (verdes)
- Todas as salas são **diferentes**

O **mapa** é uma memória externa ao circuito, de tamanho 64x64 pontos, podendo ser fornecido ao circuito diferentes mapas para localização.

O circuito é denominado **ondeestou**, e possui a seguinte interface externa:

```
entity ondeestou is
    port (
        clock:   in STD_LOGIC;
        reset:   in STD_LOGIC;
        ---- interface para pedir localização
        x, y:    in STD_LOGIC_VECTOR (5 downto 0);
        achar:   in STD_LOGIC;
        prog:    in STD_LOGIC;
        ---- interface com a memória
        address: out STD_LOGIC_VECTOR (11 downto 0);
        ponto:   in STD_LOGIC;
        ---- interface com o resultado da localização
        fim:     out STD_LOGIC;
        sala:    out STD_LOGIC_VECTOR (3 downto 0)
    );
end ondeestou;
```




Este circuito possui como **entradas** (além de **clock** e **reset**):

1. um pedido para **achar** se o par (x,y) encontra-se dentro de uma sala.
2. um pedido para **programar** o par (x,y) como a dimensão de uma sala.
3. par (**x**, **y**) correspondendo a um dado tamanho de sala ou uma dada posição no mapa.
4. uma interface de memória, correspondendo ao acesso ao mapa. O circuito fornece um endereço (**address**) e a memória (localizada no *testbench*) devolve o valor do **ponto** (‘0’ ou ‘1’).

Como resultado (**saídas**):

1. **sala**: corresponde ao número da sala de acordo com os valores definidos na configuração. Se a sala for ‘**0**’ significa que o ponto (**x**, **y**) não se encontra em nenhuma sala válida.
2. **fim**: terminou o processamento sobre o par (**x**, **y**).

A configuração das salas é enviada ao circuito **ondeestou**. Um exemplo de código VHDL para declarar o array de salas é dado abaixo (lembrando, este código deve ser inserido entre a *architecture* e o *begin*).

```
type coord is record
x:    STD_LOGIC_VECTOR (5 downto 0);
y:    STD_LOGIC_VECTOR (5 downto 0);
end record;
--- definição do array para armazenar os tamanhos das salas
constant N_SALAS: integer := 8;
type room is array(0 to N_SALAS) of coord;
signal salas : room;
```

**Antes de iniciar o processo de busca há uma etapa de programação dos tamanhos das salas pelo *testbench*. Inicialmente um contador de salas está em zero, e cada borda de subida do clock, com prog=’1’ um novo tamanho de sala é armazenada.** *Este é um processo simples, independente da máquina de estados de controle.*

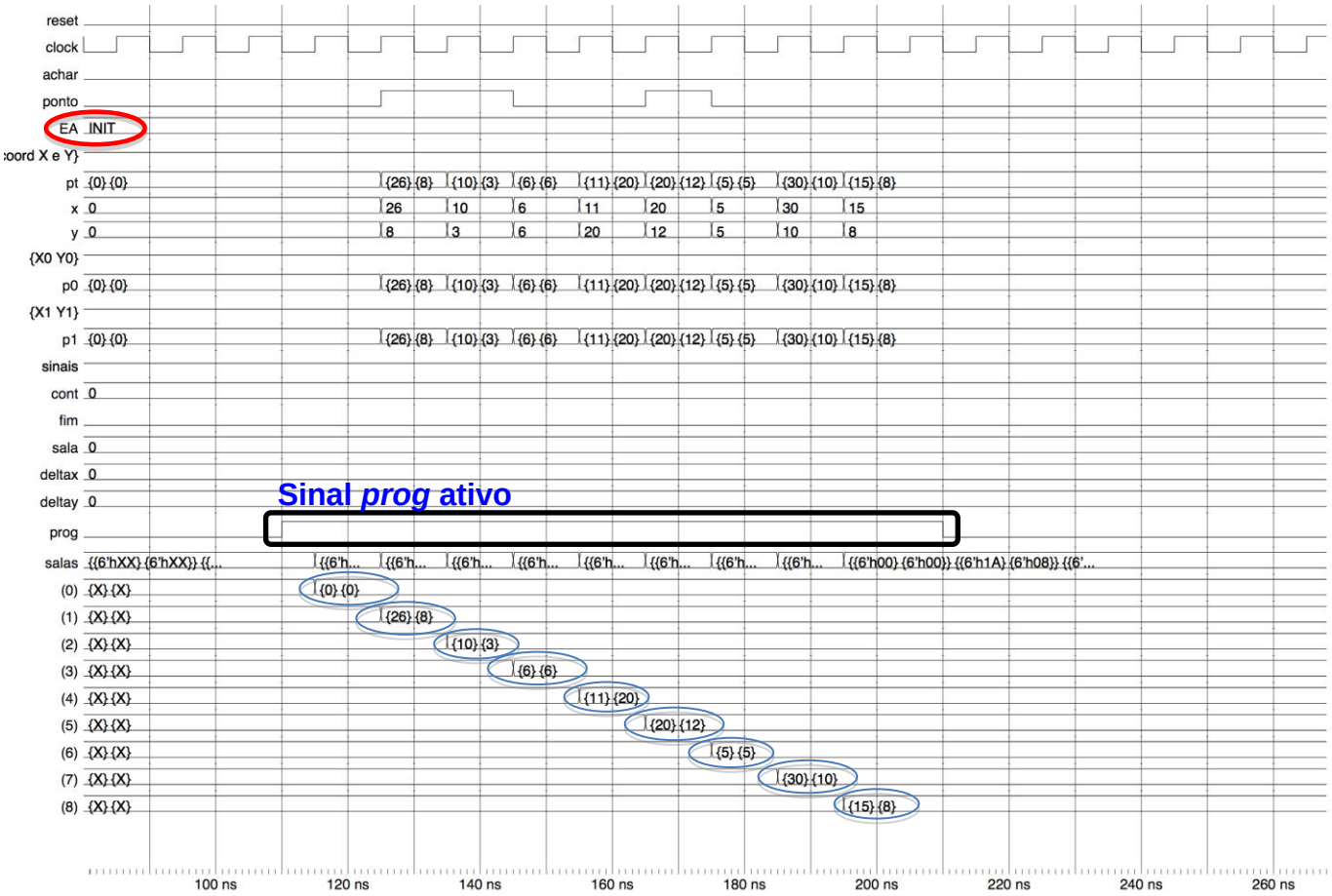
Um possível algoritmo para determinar se a posição (x,y) encontra-se um uma dada sala, e em qual sala é:

1. A partir da coordenada (x,y) incrementar y até encontrar uma borda (**Y1**).
   Se y=63 termina a execução pois chegou na fronteira superior do mapa → sala=0.
2. A partir da coordenada (x,y) decrementar y até encontrar uma borda (**Y0**).
   Se y=0 termina a execução pois chegou na fronteira inferior do mapa → sala=0.
3. A partir da coordenada (x,y) incrementar x até encontrar uma borda (**X1**).
   Se x=63 termina a execução pois chegou na fronteira direita do mapa → sala=0.
4. A partir da coordenada (x,y) decrementar x até encontrar uma borda (**X0**).
   Se x=0 termina a execução pois chegou na fronteira esquerda do mapa → sala=0.
5. Percorre **x** de (**x0**,y0) até (**x1**,y0).
   Se algum dos pontos for zero termina a execução pois não é uma linha contígua → sala=0.
6. Percorre **x** de (**x0**,y1) até (**x1**,y1).
   Se algum dos pontos for zero termina a execução pois não é uma linha contígua → sala=0.
7. Percorre **y** de (x0,**y0**) até (x0,**y1**).
   Se algum dos pontos for zero termina a execução pois não é uma linha contígua → sala=0.
8. Percorre **y** de (x1,**y0**) até (x1,**y1**).
   Se algum dos pontos for zero termina a execução pois não é uma linha contígua → sala=0.
9. $\Delta X = x1 - x0 + 1; \Delta Y = y1 - y0 + 1;$
10. Comparar o tamanho de cada sala presente nas configurações contra $(\Delta X, \Delta Y)$. Se a comparação for verdadeira, **retorna o índice da sala**.

# Exemplo de simulação que deve ser alcançada:

## (1) Etapa de programação das salas

Com **`prog='1'`** temos a programação das salas 0 a 8. A sala 0 é inválida (0,0), e as demais com coordenadas válidas (26,8) até (15,8). **Note** que a máquina de estados permanece em **`INIT`** durante a programação das coordenadas das salas.

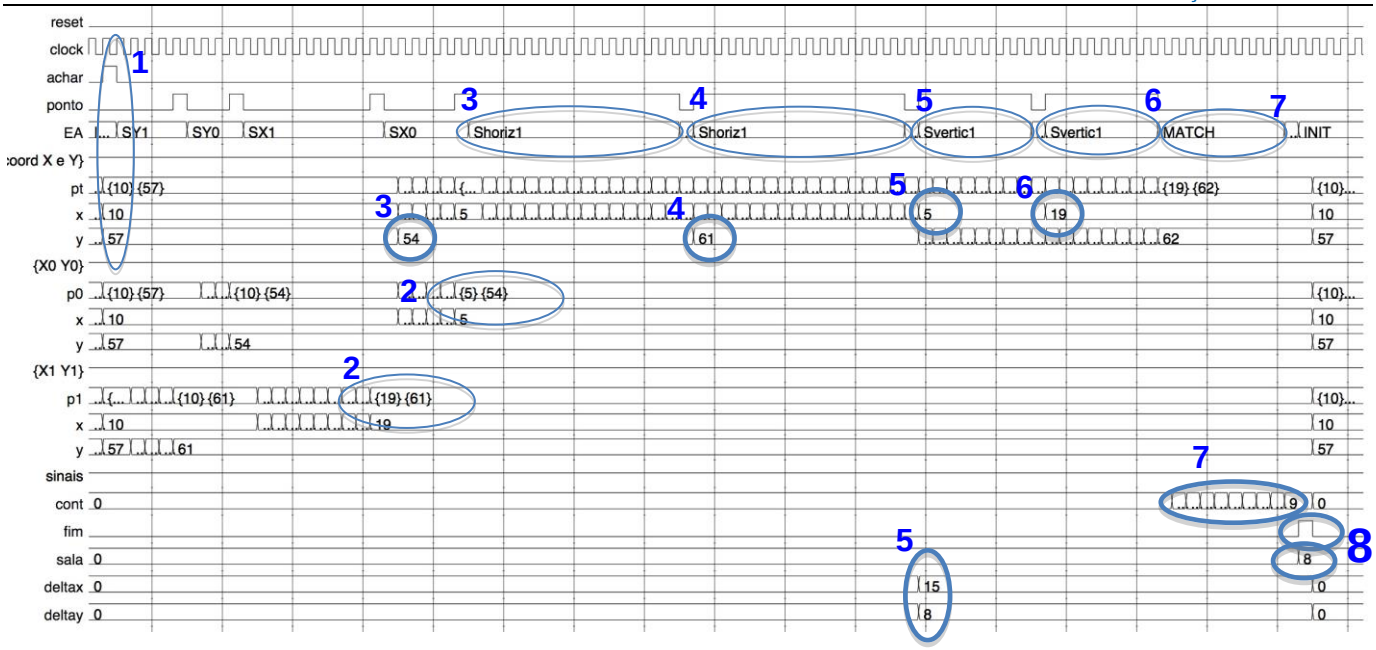



## (2) Etapa de processamento

Os eventos destacados na simulação abaixo correspondem a:

1. Par (x,y) fornecido (10, 57), e logo depois pedido para *achar* este par no mapa.
2. Após percorrer os estados SY1, SY0, SX1, SX0 as duas coordenadas de canto são encontradas: (5,54) e (19,61)
3. Primeira varredura horizontal, mantendo o Y igual a 54
4. Segunda varredura horizontal, mantendo o Y igual a 61
5. Primeira varredura vertical, mantendo o X igual a 5

   Neste estado, nesta implementação, no estado *Scoord* é feito o cálculo do deltaX e deltaY.
6. Segunda varredura vertical, mantendo o X igual a 19
7. Percorre as salas, incrementado o contador *cont*.
8. Terminou o processamento, determinou que o ponto (10,57) está na sala **8**, e sobe fim, voltando para o estado INIT.




A simulação deverá imprimir no console a seguinte sequência de mensagens:

```
ncsim> run 1 ms
** Note: Teste: 1 OK! sala: 1
#    Time: 1425 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 2 OK! sala: 0
#    Time: 1995 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 3 OK! sala: 2
#    Time: 2615 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 4 OK! sala: 0
#    Time: 3115 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 5 OK! sala: 3
#    Time: 3715 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 6 OK! sala: 0
#    Time: 4365 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 7 OK! sala: 4
#    Time: 5545 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 8 OK! sala: 0
#    Time: 6225 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 9 OK! sala: 5
#    Time: 7445 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 10 OK! sala: 0
#    Time: 7895 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 11 OK! sala: 6
#    Time: 8465 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 12 OK! sala: 7
#    Time: 9945 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 13 OK! sala: 0
#    Time: 10585 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 14 OK! sala: 8
#    Time: 11565 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 15 OK! sala: 0
#    Time: 11875 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 16 OK! sala: 5
#    Time: 13095 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 17 OK! sala: 0
#    Time: 13435 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Note: Teste: 18 OK! sala: 0
#    Time: 13965 ns  Iteration: 1  Instance: /tb_ondeestou
# ** Failure: TERMINOU O TESTE
#    Time: 13965 ns  Iteration: 1  Process: /tb_ondeestou/line__337 File: onde_estou.vhd
```

# Atividades e Avaliação

**Entrega:** um .zip contendo (aluno1aluno2.zip)

- (50% da nota) Relatório com o algoritmo utilizado para solucionar o problema e a máquina de estados implementada.
  - Explicação do algoritmo – 12,5%
  - Descrição de cada estado – 12.5%
  - Descrição de todas as condições de troca de estado – 12.5%
  - Imagem da simulação com explicação do que se observa – 12,5%
- (50% da nota) Todos os arquivos necessários para simulação
  - O código VHDL desenvolvido, comentado - 25%
  - *Testbench – 15%*
  - Script de simulação (sim.do) – 5%
  - Script de forma de onda (wave.do) – 5%

**Não haverá apresentação deste trabalho!**
**Detecção de plágio anula a nota de todos os envolvidos.**
**Não será aceito trabalho entregue fora do prazo determinado.**